Chapter 9: Virtual Memory Management

1. Background

• Virtual memory

: separation of user logical memory from physical memory

- allows only part of the program to be in memory for execution
- allows logical address space
  to be much larger than physical address space
- allows address spaces to be shared by several processes
- allows for more efficient process creation

• Virtual memory can be implemented via:

- Demand paging
- Demand segmentation

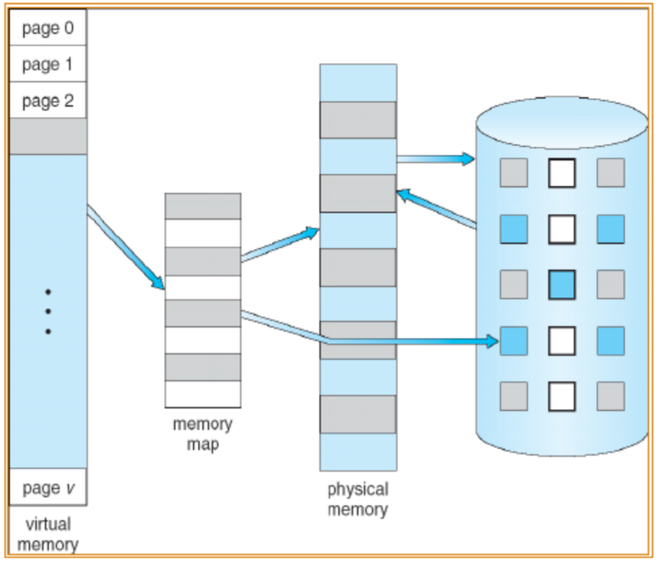
2. Virtual Memory : larger than physical memory

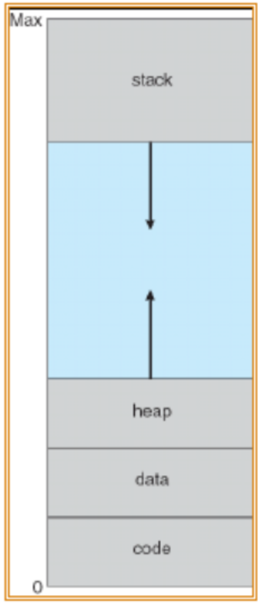
• Virtual memory involves
  the separation of logical memory and the physical memory

• The separation allows an extremely large virtual memory for programmers.

• The programmer no longer need to worry about the amount of physical memory available.

• **MMU maps virtual address to the physical address.**




3. Virtual-address Space

• Virtual address space

- The logical view of
  how process is stored in memory
- A process begins at
  • a certain logical address
  • address 0

• A process exists in contiguous memory

• A process consists of 4 segments

- Code, Data, Heap, Stack

• Heap grows upward in memory

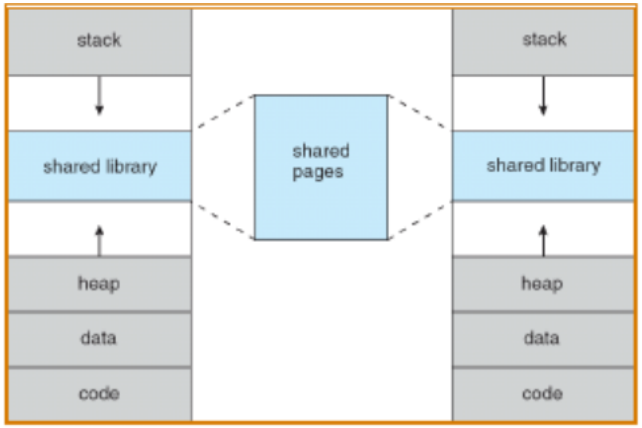
• Stack grows downward in memory

• The large blank space between the heap and the stack require actual physical pages only if the heap or stack grows.

4. Shared Library using Virtual Memory

• Virtual memory allows files and memory to be shared
  by a number of processes through page sharing.

- allows system libraries to be shared by several processes.
- allows processes to create a shared memory
- allows pages to be shard during process creation with fork()

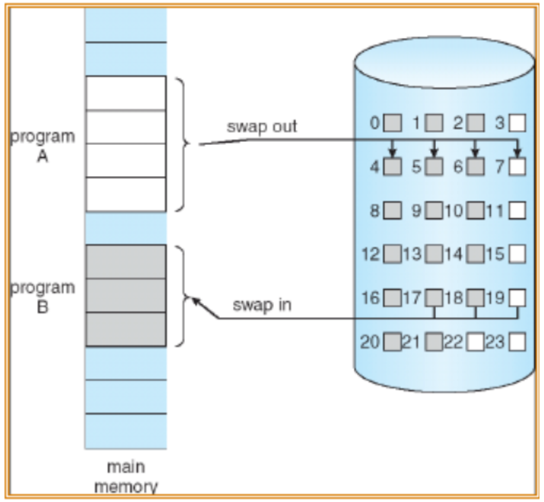


5. Demand Paging

- is a virtual memory management strategy
- Pages are loaded into memory only when they are needed during program execution → dynamic loading
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed → reference to it
  - invalid reference → abort
  - not-in-memory → bring to memory

6. Demand-paging system with swapping

- A demand-paging system is similar to a paging system with swapping where processes resides in secondary memory (disk)




- A process address space is a sequence of pages in demand paging
- When a process is created
- Lazy swapper (pager) is used
  - Never swaps a page into memory unless that page will be needed.
  - pure demand paging

* Demand paging은 swapping을 한다는 점에서 paging과 유사

* 차이: swapping할 때, 프로세스를 실행하기 위해 프로세스의 모든 페이지가 아닌, 필요한 페이지만 메모리로 불러들인다

* Swapper는 전체 프로세스를 다루지만 / Paper는 프로세스의 개별 페이지와 관련

7. Valid-Invalid Bit

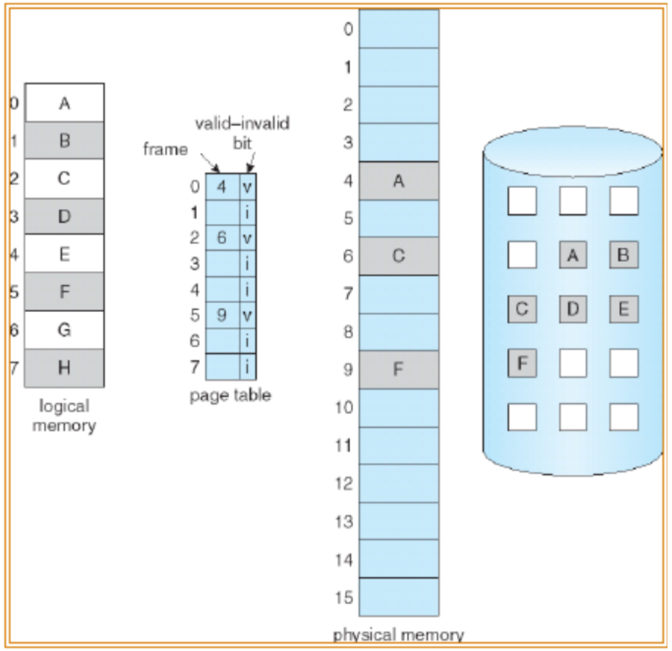
- In demand-paging system, some pages resides in the memory and others in the disk
  - needs a H/W support to distinguish it
- a valid–invalid bit is associated with each page table entry (1 → in-memory, 0 → not-in-memory)
- Initially valid – invalid bit is set to 0 on all entries
- Example of a page table snapshot:

| Frame # | valid-invalid bit |
|---|---|
| | 1 |
| | 1 |
| | 0 |
| | 1 |
| | 0 |
| | |

page table

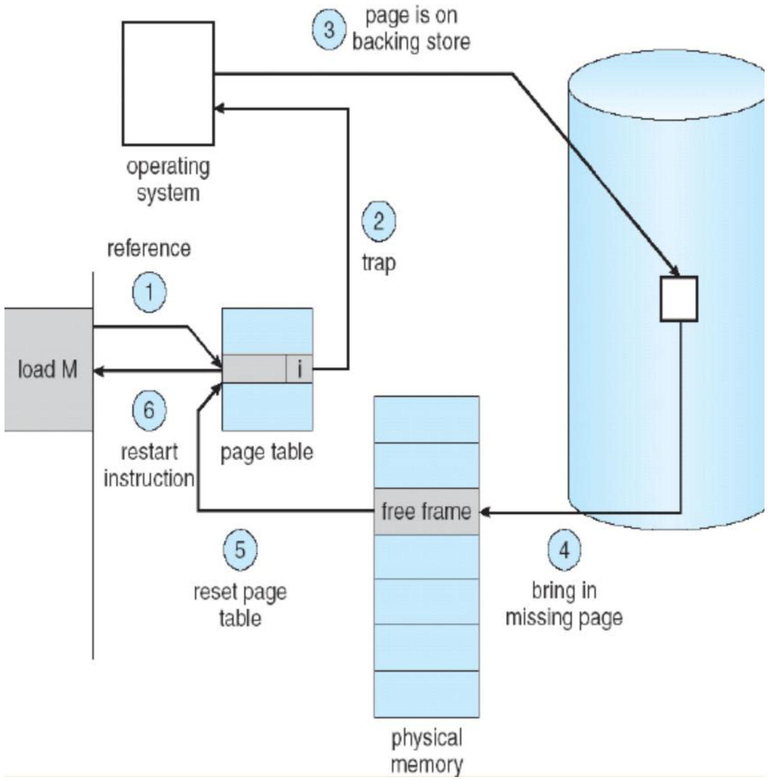
- During address translation, if valid–invalid bit in page table entry is 0 → page fault

8. Page Table when some pages aren't in main memory




- When a page is loaded in memory
  - valid-invalid bit is set :1
- When a page is in disk
  - " unset :0
  - The **frame value** contains the address of the page on the disk
- Page Fault
  - When a process tries to access a page not in memory (주소 변환을 하는 동안 페이지 테이블의 해당 엔트리의 vaid-invalid bit가 0이면)
  - Page fault trap occurs.

9. Steps in Handling a Page Fault




1. Check the reference is valid or not;
   If not, terminate the process.

1. Check the page is in memory with valid-invalid bit in page table;
→ If not,
 2. Page fault to OS
 3. Find the page on the disk
 4. Find a free frame
 4. Read the page into the newly allocated frame from disk
 5. Update the page table
 6. Restart the instruction

• Transformed into physical address
• Access physical memory to get data

* Page Fault Service Routine
1. 참조한 주소가 유효한 접근 (valid access)인지 결정하기 위해 내부 테이블(PCB)를 확인한다.
2. 유효하지 않은 접근이면 종료하고,
   유효하나 page fault이면 page in한다.
3. 비어있는 프레임을 찾는다.
4. 디스크상의 원하는 페이지의 위치를 찾아서 이를 비어있는 프레임으로 읽어드린다.
5. 해당 페이지에 대한 페이지 테이블의 엔트리를 수정한다.
6. 사용자 프로세스를 다시 시작한다.

10. What happens if there is no free frame?
• Page Fault → find a target frame for loading a new page
• What happens if there is no free frame?
 - One of the valid frame need to be replaced with the new one
• Page replacement – find a frame in memory, but not really in use,
   swap it out, swap a new page in
 - Several algorithms for page replacement
 - Performance – want an algorithm which will result in minimum number of page faults with a given reference string

11. Performance of Demand Paging
• Page Fault Rate 0 $\leq p \geq 1.0$
   - if $p = 0$ no page faults
   - if $p = 1$, every reference is a fault
• Effective Access Time (EAT)
 $EAT = (1 - p) \times$ [memory access time] $+ p \times$ [page fault time]
• Page fault time includes
 - page fault overhead - swap page out
 - swap page in  - restart overhead
Page Fault Time = page fault overhead
  = swap page out + swap page in + restart overhead

12. Demand Paging Example

• Memory access time = 200 ns (10^-9 second)

• Page fault time = 8 millisecond (10^-3 second)

• Effective Access Time (EAT)

EAT = (1 – p) x (200 nanosec) + p x (8 millisec)

= (1 – p) x (200) + p x (8,000,000)

= 200 + 7,999,800 x p (in nanosec)

• EAT is directly proportional to the page fault ratio (*p*)

13. Page Replacement

• To increase the degree of multiprogramming, we are over-allocating memory

- Σ(# of pages in each process) > # of frames in the physical memory

• In demand-paging system, only active pages reside in the physical memory

- A page is postponed to be loaded until it is used

• When a new page is loaded and there is no available frames

- One of pages in the memory needs to be replaced with the new one

=> Page Replacement

• Page Replacement completes separation between logical memory and physical memory

- large virtual memory can be provided on a smaller physical memory

14. Need For Page Replacement

• Physical memory

- Total 8 frames

- 2 of them for OS

- 6 are used for users

• Two user processes

- With 4 pages each

- 3 of them loaded

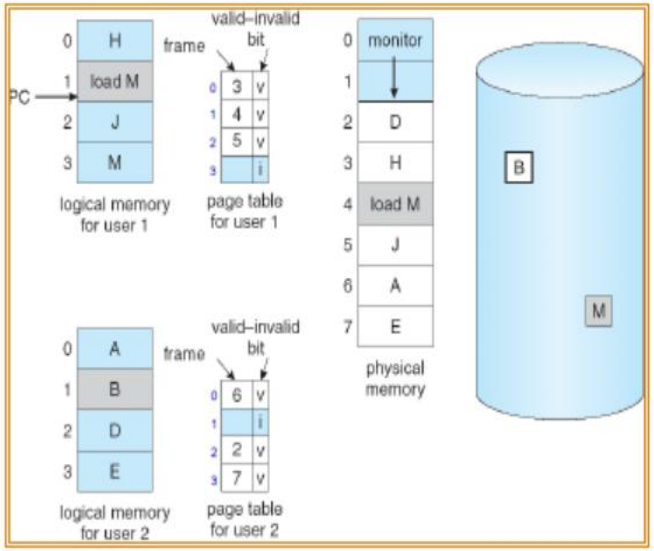
• All frames are used

• PC indicates an instruction:

- Load M

- Page M is on the disk

• Page Replacement needed




15. Basic Page Replacement

1. Find the location of the desired page on disk

2. Find a free frame:

- If there is a free frame, use it

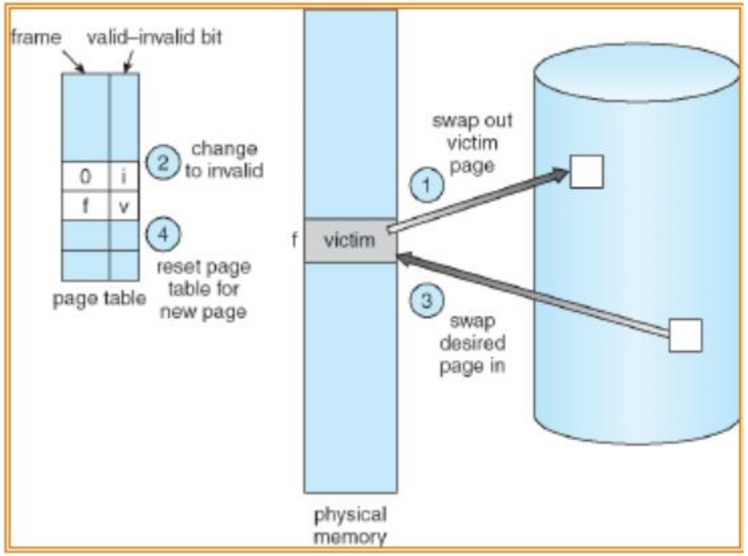
- If there is no free frame,

use a page replacement algorithm to select a victim frame

3. Read the desired page into the (newly) free frame and update the page and frame tables

4. Restart the process

16. Page Replacement procedure




• After finding the target page on the disk and the victim page using page replacement algorithm

1. Swap out the victim page
2. Unset the valid-invalid bit of the victim page
3. Swap the desired page in
4. Set the valid-invalid bit of the target page

17. Page Replacement Algorithms

• Many page replacement algorithm

- FIFO page replacement
- Optimal page replacement
- LRU page replacement
- LRU-approximation page replacement
  • Additional-Reference-Bits algorithm
  • Second-Chance algorithm
- Counting-based page replacement

• Performance metric for page replacement algorithm

- The page fault rate
- The lower the page-fault rate, the better the performance

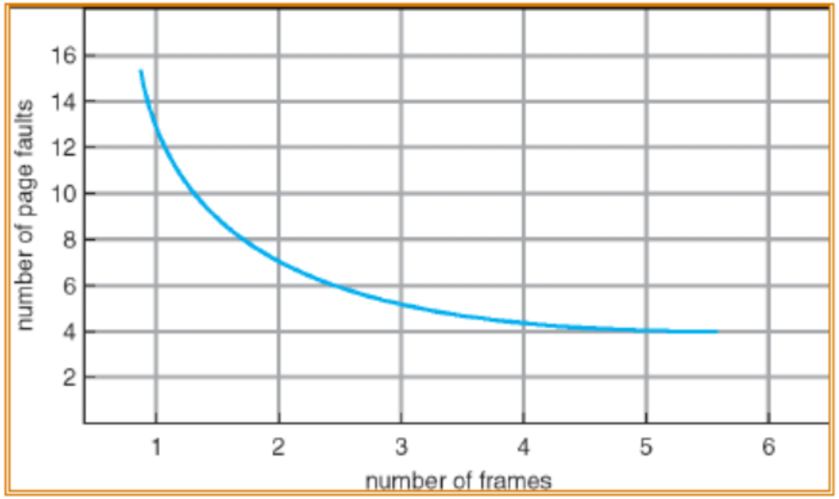
• Evaluate algorithm by

- running it on a particular string of memory references (reference string) and
- computing the number of page faults on that string

• An example of a reference string is [1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5]

18. Page Faults vs. # of Frames

• 프레임 수 많아지만 frame fault 덜 일어난다




19. 알고리즘들 생략

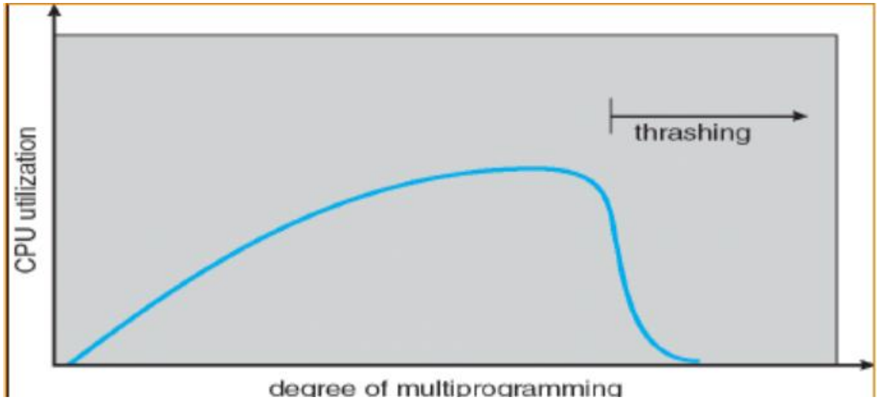
해보긔

30. Global vs. Local Allocation
• Global replacement – process selects a replacement frame from the set of all frames
  - one process can take a frame from another
• Local replacement – each process selects from only its own set of allocated frames

31. Thrashing
• If a process does not have “enough” frames,
  the pagefault rate is very high. This leads to:
- CPU utilization가 너무 낮으면
- os가 새로운 프로세스를 시스템에 추가해서 멀티프로그래밍의 degree를 높인다.

• Thrashing : a process is busy in swapping pages in and out
- High paging activity
- A process is thrashing if it is spending more time paging than executing
페이지 폴트가 너무 많이 발생, 실행보다 페이징에 시간 더걸림

32. Thrashing (Cont.)
• CPU utilization vs. degree of multiprogramming (DM)
  - As DM increases,
CPU utilization also increases, until the maximum is reached.
  - As DM is increased even further, thrashing sets in, and CPU utilization drops sharply.
- Decreasing DM can be a solution for the thrashing




33. Demand Paging and Thrashing
• To prevent thrashing, we must provide a process a proper number of frames in demand paging scheme. (프로세스에게 필요로 하는 많은 프레임을 제공해야함)
  - How do we know how many frames it need?
  - Working-Set model with locality is a solution.

• Locality model
- As a process executes, it moves from locality to locality
  - A locality is a set of pages that are actively used together.
  - A program is generally composed of several different localities
  - The localities may overlap
• Why does thrashing occur?
  ∑ size of locality > total memory size

34. Locality in a Memory-Reference Pattern
• At a certain period of execution time.
  - Only a set of pages are actively referenced
  - Others are not accessed
• The active pages (locality) migrates from one to another as time goes on.
• A program has several different localities
• Localities may overlap

## 35. Page-Fault Frequency Scheme

• Use the page-fault frequency (PFF) to determine the proper number of frames for a process

• Establish "acceptable" page-fault rate

- If page-fault rate too low, process loses frame
- If page-fault rate too high, process gains frame

acceptable rate

## 36. Other Issues - Prepaging

- to reduce the large number of page faults that occurs at process startup
- prepage all or some of the pages a process will need, before they are referenced
- But if prepaged pages are unused, I/O and memory is wasted
- Assume $s$ pages are prepaged and $\alpha$ ($0 \leq \alpha \leq 1$) of the pages is used

• Is the cost of $s * \alpha$ saving page faults > or < than the cost of prepaging $s * (1 - \alpha)$ unnecessary pages?

• $\alpha$ close to 0 –> prepaging loses

• $\alpha$ close to 1 –> prepaging wins

## 37. Other Issues – Page Size

• Page size selection must take into consideration:

- Fragmentation (internal)
  - page size 사이즈가 크면 내부 단편화 발생
- Page table size
  - page size 줄이면 page 수가 늘어남
- I/O overhead
  - page size가 크면 I/O transfer time이 길어짐
- Locality

• page size가 크면 locality를 만족시키는 페이지 수가 줄어듦

## 38. Other Issues – TLB Reach

• TLB Reach - The amount of memory accessible from the TLB

- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB
  Otherwise there is a high degree of TLB miss

• How to increase the TLB Reach

- Increase the Page Size.
- Increase the TLB size

• Increase the Page Size

- This may lead to an increase in fragmentation as not all applications require a large page size

• Provide Multiple Page Sizes

- 8KB, 64 KB, 512 KB, 4 MB in Sun UltraSPARC OS
- This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation

## 39. Other Issues – I/O interlock

• Pages must sometimes be locked into memory

• Consider I/O - I/O Interlock

- Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm.

• Lock bit

- Associated with each page
- Set to indicate that the page is locked
- Can avoid to be selected by page replacement algorithm